

(long) An AES implementation for 32-bit platforms

Source: <http://www.derkeiler.com/Newsgroups/sci.crypt/2003-06/0236.html>

From: Mok-Kong Shen (mok-kong.shen_at_t-online.de)

Date: 06/10/03

Date: Tue, 10 Jun 2003 20:55:48 +0200

Recently in discussions in another thread I suggested that unions are well suited for byte-array implementations of AES. For with a definition like

```
union BLOCK
{ byte b[16];
  byte bm[4][4];
  word w[4];
};
BLOCK yy,zz;
```

yy.b is a byte array of size 16 and yy.bm is a 4*4 byte matrix, where e.g. the elements of a row such as yy.bm[2] may be permuted (thru permuting the individual elements yy.bm[2][i]), and, e.g. yy.w[0] is a word (a container for the four bytes yy.b[0] to yy.b[3]) which can be moved as a single entity (with one operation) or xor-ed with another word e.g. zz.w[3], resulting under a single xor operation the xor-ing of the byte yy.b[0] with zz.b[12], byte yy.b[1] with zz.b[13], byte yy.b[2] with zz.b[14] and byte yy.b[3] with zz.b[15] (i.e. instead of four individual operations). Note that no interpretation of the content of the word as an integer is ever needed in the AES algorithm. Thus, since we don't utilize the '<<' and the '>>' operator, the difference of endian-ness (in the proper sense, not 'bit sex') of hardware is of no relevance to us.

In fact, in my view this advantage apparently underlies an implementation scheme of AES for 32-bit platforms originally due to Daemen and Rijmen. See their book p.56ff. (W. Stallings' paper in Cryptologia, July 2002 also describes it but has some typos (errors in indices)).

Since from the said previous discussions and a search on the internet I gained the strong impression that

sci.crypt: (long) An AES implementation for 32-bit platforms

such an implementation was not previously undertaken, I made an effort to do one. The result is the code in the attachment below. The code should be easily readable with the help of FIPS-197 and one of the literatures mentioned above. It consists of two parts. The first part generates some tables for later runs of the second part which performs the actual AES processing. A demo program for the testvectors in FIPS is also included and has been successfully run under VC++.

There seems to be, to my surprise, very few publically (freely) available AES implementations. I should appreciate it very much, if some readers would like to make a comparison between my implementation and the well-established ones, either in aspects of efficiency (which is certainly of primary importance) or of readability.

Thanks in advance and sincere apology for the very long post.

M. K. Shen

```
// An AES implementation for 32-bit platforms
// Consisting of Part 1 and Part 2
// Version 1.0
// Release date: 10th June 2003
// Author: Mok-Kong Shen, Munich, Germany

// Part 1 of the AES package

// Installation program to generate some tables in a file
// for later use in encryption processing (see Part 2).
// The directory c:\aes\ is assumed to exist (change that
// in the code below, similarly in Part 2, if necessary).
// See copyright information at the end of this package.

#include <stdio.h>

typedef unsigned char byte;
typedef unsigned int word;

byte sbox[256]={
0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,
0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76,
0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,
0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0,
0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,
0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15,
0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,
```

(long) An AES implementation for 32-bit platforms

sci.crypt: (long) An AES implementation for 32-bit platforms

```
0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75,  
0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,  
0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84,  
0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,  
0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf,  
0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,  
0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8,  
0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,  
0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2,  
0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,  
0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73,  
0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,  
0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb,  
0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,  
0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79,  
0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,  
0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08,  
0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,  
0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a,  
0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,  
0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e,  
0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,  
0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf,  
0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,  
0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16};
```

```
byte invsbox[256];
```

```
byte gfmult(byte a, byte b)
```

```
{ byte r,t;  
  if (a<b) { t=b; b=a; a=t; }  
  r=0;  
  do  
  { if (b&0x01) r^=a;  
    if (a&0x80) a=(a<<1)^0x1b; else a<<=1;  
  } while (b>>=1);  
  return r;  
}
```

```
union TABLE
```

```
{ byte b[1024];  
  byte bm[256][4];  
  word w[256];  
};
```

```
// The tables t's and v's are for encryption and decryption  
// respectively (see the book of Daemen and Rijmen, p.56ff)
```

```
TABLE t0,t1,t2,t3,v0,v1,v2,v3;
```

sci.crypt: (long) An AES implementation for 32-bit platforms

```
void computetables()
{ int i,j;
  byte s,r;
  for (i=0; i<256; i++)
  { s=sbox[i];
    t0.bm[i][0]=gfmult(s,0x02); t0.bm[i][1]=t0.bm[i][2]=s;
    t0.bm[i][3]=gfmult(s,0x03);
    r=invSbox[i];
    v0.bm[i][0]=gfmult(r,0x0e); v0.bm[i][1]=gfmult(r,0x09);
    v0.bm[i][2]=gfmult(r,0x0d); v0.bm[i][3]=gfmult(r,0x0b);
    for (j=0; j<4; j++)
    { t1.bm[i][j]=t0.bm[i][(j+3)%4];
      t2.bm[i][j]=t0.bm[i][(j+2)%4];
      t3.bm[i][j]=t0.bm[i][(j+1)%4];
      v1.bm[i][j]=v0.bm[i][(j+3)%4];
      v2.bm[i][j]=v0.bm[i][(j+2)%4];
      v3.bm[i][j]=v0.bm[i][(j+1)%4];
    }
  }
}
```

```
void main()
{ FILE *fp;
  int i;
  for (i=0; i<256; i++) invSbox[sbox[i]]=i;
  computetables();
  // Note that c:/aes/tables.bin corresponds to c:\aes\tables.bin
  // of Microsoft OS
  if (!(fp=fopen("c:/aes/tables.bin","wb"))) goto error;
  if (( fwrite(sbox,256,1,fp)!=1 ||
        fwrite(invSbox,256,1,fp)!=1 ||
        fwrite(t0.b,1024,1,fp)!=1 || fwrite(t1.b,1024,1,fp)!=1 ||
        fwrite(t2.b,1024,1,fp)!=1 || fwrite(t3.b,1024,1,fp)!=1 ||
        fwrite(v0.b,1024,1,fp)!=1 || fwrite(v1.b,1024,1,fp)!=1 ||
        fwrite(v2.b,1024,1,fp)!=1 || fwrite(v3.b,1024,1,fp)!=1 )
      || ferror(fp) ) goto error;
  fclose(fp); printf("tables successfully installed\n");
  return;
error:
  printf("error in writing tables. installation failed\n");
  fclose(fp);
}
```

```
// Part 2 of the AES package (for normal use after Part 1
// has been run)
```

```
// AES encryption/decryption functions and a main program
// demonstrating the processing of the testvectors in FIPS-197.
// See copyright information at the end of this package.
```

sci.crypt: (long) An AES implementation for 32-bit platforms

```
#include <stdio.h>
#include <stdlib.h>

typedef unsigned char byte;
typedef unsigned int word;

union UU
{ byte b[4];
  word w;
};

union BLOCK
{ byte b[16];
  byte bm[4][4];
  word w[4];
};

BLOCK aesin,aesout;

const int Nb=4;

int Nk,Nr;

int ekeywdim;
int setupdone=0;

byte sbox[256],invsbox[256];

// Maximum storage space is provided for ukey and ekey,
// the actual size in words is Nk in ukey and ekeywdim
// in ekey

union USERKEY
{ byte b[32];
  byte bm[8][4];
  word w[8];
} ukey;

union EXPANDEDKEY
{ byte b[240];
  byte bm[60][4];
  word w[60];
} ekey;

union TABLE
{ byte b[1024];
  byte bm[256][4];
  word w[256];
};
```

sci.crypt: (long) An AES implementation for 32-bit platforms

```
// The tables t's and v's are for encryption and decryption
// respectively (see the book of Daemen and Rijmen, p.56ff).
```

```
TABLE t0,t1,t2,t3,v0,v1,v2,v3;
```

```
// Before calling aassetup, contents of the tables must be
// available in the file designated. See Part 1 of the package.
```

```
void aassetup(int keylength)
```

```
{ FILE *fp;
// Note that c:/aes/tables.bin corresponds to
// c:\aes\tables.bin of Microsoft OS
if (!(fp=fopen("c:/aes/tables.bin","rb"))) goto error;
if (( fread(sbox,256,1,fp)!=1 || fread(invbox,256,1,fp)!=1 ||
    fread(t0.b,1024,1,fp)!=1 || fread(t1.b,1024,1,fp)!=1 ||
    fread(t2.b,1024,1,fp)!=1 || fread(t3.b,1024,1,fp)!=1 ||
    fread(v0.b,1024,1,fp)!=1 || fread(v1.b,1024,1,fp)!=1 ||
    fread(v2.b,1024,1,fp)!=1 || fread(v3.b,1024,1,fp)!=1 )
    || ferror(fp) ) goto error;
goto normal;
error:
printf("error in reading tables from file\n"); fclose(fp);
exit(1);
normal:
fclose(fp);
switch(keylength)
{ case 128: Nk=4; Nr=10; break;
  case 192: Nk=6; Nr=12; break;
  case 256: Nk=8; Nr=14; break;
  default: printf("wrong keylength in aassetup call\n");
           exit(1);
}
ekeywdim=Nb*(Nr+1);
setupdone=1;
}
```

```
const byte RC[12]={ 0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
                   0x40, 0x80, 0x1b, 0x36, 0x6c };
```

```
void userkeyexpansion()
```

```
{ UU temp;
  byte t;
  int i;
// See FIPS-197
for (i=0; i<Nk; i++) ekey.w[i]=ukey.w[i];
for (i=Nk; i<ekeywdim; i++)
{ temp.w=ekey.w[i-1];
  if ((i%Nk)==0)
  { t=temp.b[0]; temp.b[0]=sbox[temp.b[1]]^RC[i/Nk];
    temp.b[1]=sbox[temp.b[2]]; temp.b[2]=sbox[temp.b[3]];
    temp.b[3]=sbox[t];
  }
}
```

sci.crypt: (long) An AES implementation for 32-bit platforms

```

} else if (Nk>6 && (i%Nk)==4)
{ temp.b[0]=sbox[temp.b[0]]; temp.b[1]=sbox[temp.b[1]];
  temp.b[2]=sbox[temp.b[2]]; temp.b[3]=sbox[temp.b[3]];
}
ekey.w[i]=ekey.w[i-Nk]^temp.w;
}
}

```

```

void ekeyinvmixcolumnstransform()
{ const byte matrix[4][4]=
  { { 0x0e, 0x0b, 0x0d, 0x09 }, { 0x09, 0x0e, 0x0b, 0x0d },
    { 0x0d, 0x09, 0x0e, 0x0b }, { 0x0b, 0x0d, 0x09, 0x0e } };
  UU temp;
  byte rr,r,a,b;
  int ii,i,j,ii1,ii2;
// The first and the last round key are excluded from the
// transform
ii1=4; ii2=4*Nr-1;
for (ii=ii1; ii<=ii2; ii++)
{ temp.w=ekey.w[ii];
  for (i=0; i<4; i++)
  { rr=0;
    for (j=0; j<4; j++)
    { a=matrix[i][j]; b=temp.b[j]; r=0;
      do // GF multiplication of a and b
      { if (b&0x01) r^=a;
        if (a&0x80) a=(a<<1)^0x1b; else a<<=1;
      } while (b>>=1);
      rr^=r;
    }
    ekey.bm[ii][i]=rr;
  }
}
}

```

```
enum PROCESS { encrypt, decrypt } encdec;
```

```

// Before calling aeskeyschedule, key of the user must be
// available in ukey in the bytes from ukey.b[0] to
// ukey.b[4*Nk-1].
// It is assumed that aassetup has been called previously.

```

```

void aeskeyschedule(PROCESS kind)
{ if (!setupdone)
{ printf("error: aassetup was not called\n"); exit(1); }
  encdec=kind;
  userkeyexpansion();
// See the book of Daemen and Rijmen, p.59.
  if (encdec==decrypt) ekeyinvmixcolumnstransform();
}

```

sci.crypt: (long) An AES implementation for 32-bit platforms

```
// Before calling aesprocess, input data must be available
// in aesin (bytes aesin.b[0] to aesin.b[15]), output will
// be in aesout (bytes aesout.b[0] to aesout.b[15])
// It is assumed that aeskeyschedule has been called previously.
```

```
void aesprocess()
{ BLOCK temp;
  int j,grd,ekeyw0;
  if (encdec==encrypt)
  { for (j=0; j<4; j++) aesout.w[j]=aesin.w[j]^ekey.w[j];
    ekeyw0=4;
    for (grd=1; grd<Nr; grd++)
    { for (j=0; j<4; j++) temp.w[j]=aesout.w[j];
      for (j=0; j<4; j++) aesout.w[j]=t0.w[temp.bm[j][0]]^
        t1.w[temp.bm[(j+1)%4][1]]^t2.w[temp.bm[(j+2)%4][2]]^
        t3.w[temp.bm[(j+3)%4][3]]^ekey.w[ekeyw0+j];
      ekeyw0+=4;
    }
    for (j=0; j<4; j++) temp.w[j]=aesout.w[j];
    for (j=0; j<4; j++)
    { aesout.bm[j][0]=
      sbox[temp.bm[j][0]]^ekey.bm[ekeyw0][0];
      aesout.bm[j][1]=
      sbox[temp.bm[(j+1)%4][1]]^ekey.bm[ekeyw0][1];
      aesout.bm[j][2]=
      sbox[temp.bm[(j+2)%4][2]]^ekey.bm[ekeyw0][2];
      aesout.bm[j][3]=
      sbox[temp.bm[(j+3)%4][3]]^ekey.bm[ekeyw0][3];
      ekeyw0++;
    }
  }
  else
  { ekeyw0=Nb*Nr;
    for (j=0; j<4; j++) aesout.w[j]=aesin.w[j]^ekey.w[ekeyw0+j];
    ekeyw0-=4;
    for (grd=1; grd<Nr; grd++)
    { for (j=0; j<4; j++) temp.w[j]=aesout.w[j];
      for (j=0; j<4; j++) aesout.w[j]=v0.w[temp.bm[j][0]]^
        v1.w[temp.bm[(j+3)%4][1]]^v2.w[temp.bm[(j+2)%4][2]]^
        v3.w[temp.bm[(j+1)%4][3]]^ekey.w[ekeyw0+j];
      ekeyw0-=4;
    }
    for (j=0; j<4; j++) temp.w[j]=aesout.w[j];
    for (j=0; j<4; j++)
    { aesout.bm[j][0]=
      invsbox[temp.bm[j][0]]^ekey.bm[ekeyw0][0];
      aesout.bm[j][1]=
      invsbox[temp.bm[(j+3)%4][1]]^ekey.bm[ekeyw0][1];
      aesout.bm[j][2]=
      invsbox[temp.bm[(j+2)%4][2]]^ekey.bm[ekeyw0][2];
      aesout.bm[j][3]=
```

sci.crypt: (long) An AES implementation for 32-bit platforms

```

    invsbox[temp.bm[(j+1)%4][3]]^ekey.bm[ekeyw0][3];
    ekeyw0++;
}
}
}

// Display an byte array. For test purposes only.

void display(byte b[], int size)
{ int i,k=0;
  for (i=0; i<size; i++)
  { printf(" %2.2x",b[i]); if ((i%16)==15) printf("\n"); }
  if ((size%16)!=0) printf("\n"); printf("\n");
}

// Demo main program

void main()
{ int i,keylength;

// plaintext of testvectors in FIPS (same in all cases)
const byte pt[16]={ 0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,
                   0x88,0x99,0xaa,0xbb,0xcc,0xdd,0xee,0xff};
// user-key of testvectors for keylength 256
// the first part (16/24 bytes) is used for keylength 128/192
const byte kg[32]={ 0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,
                   0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,
                   0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,
                   0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f};

// setting ukey (array is used in all cases of keylength)
for (i=0; i<32; i++) ukey.b[i]=kg[i];

label:

printf("enter keylength (128/192/256) (0 for termination):\n");
scanf("%d",&keylength);
if (keylength==0) { printf("demo run ended\n"); return; }
aessetup(keylength);
printf("\nuser-key:\n"); display(ukey.b,Nb*Nk);

aeskeyschedule(encrypt);
for (i=0; i<16; i++) aesin.b[i]=pt[i];
printf("plaintext:\n"); display(aesin.b,16);
aesprocess();
printf("ciphertext:\n"); display(aesout.b,16);

aeskeyschedule(decrypt);
for (i=0; i<4; i++) aesin.w[i]=aesout.w[i];
printf("decryption input:\n"); display(aesin.b,16);
aesprocess();

```

sci.crypt: (long) An AES implementation for 32-bit platforms

```
printf("plaintext recovered:\n"); display(aesout.b,16);
```

```
goto label;  
}
```

```
// Copyright (c) Mok-Kong Shen 2003. mok-kong.shen@t-online.de  
// This work (consisting of Part 1 and Part 2) and all modified  
// versions of it may be freely copied, modified and used for  
// all legal civilian purposes subject to the following  
// conditions:  
// (1) A copy of this copyright notice with the modification  
// history list must be included in any copy of this work  
// or any modified versions of it.  
// (2) Any modifications (excepting the function main of  
// part 2 and the name of the file storing the tables)  
// should be documented in the modification history list  
// below with appropriate descriptions and date and name  
// and address of the person performing the modifications.  
// (3) In case of non-trivial modifications, i.e. those  
// arising from efficiency or correctness issues, a copy  
// of the modified package is to be sent to the copyright  
// owner at the address above.  
// (4) Eventual negative consequences in connection with the  
// use of this work or its modified versions do not  
// constitute any liabilities on the part of the copyright  
// owner.  
// Any other usages of the contents of the package require  
// prior permission from the copyright owner.  
//  
// Modification history list:  
// (currently empty)  
//  
//  
//
```